

APPENDIX B

EXAMPLE ROM SCRIPTS FOR RISC PROCESSOR CORE OF IPCM

9.13 Example Scripts (ROM contents)

9.13.1 Vectors

The first 32 addresses are jumps to routines (reset at address 0, illegal instruction at address 1, etc.)

```
0    8051 // start: jmp GetCh0PC
1    8001 // 1:      jmp 1b          // 1 - illegal instruction trap
2    80fc //           jmp MDdma     // 2 - copy from MCore to DSP
3    8150 //           jmp DMdma     // 3 - copy from DSP to MCore
4    0088 //           NOP (mov r0,r0)
5    0088 //
6    0088 //
7    0088 //
8    0088 //
9    0088 //
10   0088 //
11   0088 //
12   0088 //
13   0088 //
14   0088 //
15   0088 //
16   0088 //
17   0088 //
18   0088 //
19   0088 //
```

```

20 0088 // NOP
21 0088 // NOP
22 0088 // NOP
23 0088 // NOP
24 0088 // NOP
25 0088 // NOP
26 0088 // NOP
27 0088 // NOP
28 0088 // NOP
29 0088 // NOP
30 0088 // NOP
31 0088 // NOP

```

9.13.2 Context Switch

The following code is the context switch routine in ROM; notice that almost all the instructions here cannot work anywhere else; the execution order is the only possible one.

32 00e3 //	CtxPtrInit	// DM[0x7002] --> r0
33 01e3 //	CatchCPtr	// address to spill registers
34 00e0 //	ldMAstG1	// (GReg[1] saved) & (MA --> GReg[1])
35 01e0 //	ldMDstG2	// (GReg[2] saved) & (MD --> GReg[2])
36 02e0 //	ldMSstG3	// (GReg[3] saved) & (MS --> GReg[3])
37 03e0 //	ldDAstG4	// (GReg[4] saved) & (DA --> GReg[4])
38 04e0 //	ldDDstG5	// (GReg[5] saved) & (DD --> GReg[5])
39 05e0 //	ldDSstG6	// (GReg[6] saved) & (DS --> GReg[6])
40 02e3 //	stG7mvShPC	// (GReg[7] s.) & (ShPC --> GReg[7])
41 03e3 //	stG7mvShLoop	// (ShPC saved) & (ShLoop --> GReg[7])
42 07e0 //	stG7ldCS	// (ShLoop saved) & (CS --> GReg[7])
43 00e0 //	ldMAstG1	// (MA saved)
44 01e0 //	ldMDstG2	// (MD saved)
45 02e0 //	ldMSstG3	// (MS saved)
46 03e0 //	ldDAstG4	// (DA saved)
47 04e0 //	ldDDstG5	// (DD saved)
48 05e0 //	ldDSstG6	// (DS saved)
49 06e0 //	ldCAstG7	// (CS saved)
50 01e2 //	stCAmovShReg02Gr1	
51 00e3 //	CtxPtrInit	
52 01e3 //	CatchCPtr	
53 01e4 //	TstPendingAndSwitch// ShReg0 saved	
54 00e3 //	CtxPtrInit	// DM[0x7002] --> GReg[0]
55 01e3 //	CatchCPtr	// Get address for restoring registers
56 1811 //	addi r0,#17	// Points to CA of channel to restore
57 04e3 //	ldFU0inLd0	
58 05e3 //	mvFU02G1	// GReg[1] = CA
59 07e1 //	ldmfub7	// CA restored & fetch CS
60 06e1 //	ldmfub6	// CS restored & fetch DS
61 05e1 //	ldmfub5	// DS restored & fetch DD
62 04e1 //	ldmfub4	// DD restored & fetch DA
63 03e1 //	ldmfub3	// DA restored & fetch MS
64 02e1 //	ldmfub2	// MS restored & fetch MD
65 01e1 //	ldmfub1	// MD restored & fetch MA

```

66  00e1 //      ldmfub0    // MA restored & fetch ShLoop
67  06e3 //      ldShLoop   // ShLoop restored & fetch ShPC
68  07e3 //      ldShPC     // ShPC restored & fetch GReg[7]
69  07e2 //      ldmGReg7  // GReg[7] restored & fetch GReg[6]
70  06e2 //      ldmGReg6  // GReg[6] restored & fetch GReg[5]
71  05e2 //      ldmGReg5  // GReg[5] restored & fetch GReg[4]
72  04e2 //      ldmGReg4  // GReg[4] restored & fetch GReg[3]
73  03e2 //      ldmGReg3  // GReg[3] restored & fetch GReg[2]
74  02e2 //      ldmGReg2  // GReg[2] restored & fetch GReg[1]
75  00e2 //      ldmGReg1GReg0 // GReg[1] & GReg[0] restored
76  00e4 //      cpshReg    // copy all shadow registers (PC, ...)
77  0088 //      NOP
78  0088 //      NOP
79  0088 //      NOP

```

9.13.3 Boot Code

The standard channel 0 boot code.

```

80  80ba //      jmp Reset      // jump around common subroutines
81  0870 //      GetCh0PC:ldi r0,REGPAGE
82  0011 //      revblo r0      // r0 = 0x7000
83  51e0 //      ld rT,(r0,CHNOADDR) // load first pc value of channel 0
84  0108 //      jmpr rT
85  0300 // 0:      done 3      //clear HE set HI
86  0a70 // Mccb:    ldi rC,REGPAGE
87  0211 //      revblo rC      // pointer to register page
88  501a //      ld r0,(rC,R_CCR) // load current channel number
89  5202 //      ld rC,(rC,R_MC0PTR) // load channel 0 ptr
90  4a00 //      cmpeqi rC,0
91  7df9 //      bt 0b
92  0017 //      lsll r0
93  0017 //      lsll r0
94  0017 //      lsll r0
95  0017 //      lsll r0
96  0298 //      add rC,r0
97  0006 // 0:      ret
98  0700 // 0:      done 7      // clear DE set DI
99  0a70 // Dccb:    ldi rC,REGPAGE
100 0211 //      revblo rC      // pointer to register page
101 501a //      ld r0,(rC,R_CCR) // load current channel number
102 520a //      ld rC,(rC,R_DC0PTR) // load channel 0 ptr
103 4a00 //      cmpeqi rC,0
104 7df9 //      bt 0b
105 0017 //      lsll r0
106 0017 //      lsll r0
107 0017 //      lsll r0
108 0017 //      lsll r0
109 0298 //      add rC,r0
110 0006 // 0:      ret
111 0007 // MDone:    clrf 0      // clear flags
112 6b00 //      stf rP,ma    // address of status word

```

```

113 3501 // andni rS,BD_DONE // indicate complete
114 6d12 // stf rS,md,sz16 // update BD flags (DO NOT FLUSH!)
115 6e16 // stf rN,md,sz16,fls// update BD count
116 7f18 // bdf Mcerr
117 4508 // tsti rS,BD_INTR // should we interrupt?
118 7c01 // bf Of
119 0201 // notify 2 // yes, do it
120 1b08 // 0: addi rP,BD_SIZE // bump ptr, assume no wrap
121 4502 // tsti rS,BD_WRAP // are we really at the end
122 7c06 // bf lf // if not, skip the following
123 0007 // MFIRST: clrf 0 // clr flags, could enter here
124 038a // mov rP,rC
125 1b04 // addi rP,CB_BPTR // offset to ptr to BD array
126 6b04 // stf rP,ma,pre
127 6313 // ldf rP,md,sz32 // get BD array pointer
128 7e0c // bsf Mcerr
129 0006 // 1: ret
130 0300 // Mblock: done 3 // must wait on MCORE clear HE set HI
131 0007 // MGetBD: clrf 0 // make sure flags are clear
132 6b04 // stf rP,ma,pre // new DMA address to load from
133 6516 // ldf rS,md,sz16,pre// get status from buf desc
134 7e06 // bsf Mcerr
135 4501 // tsti rS,BD_DONE // check for done bit
136 7cf9 // bf Mblock // if zero, not a valid BD
137 6616 // ldf rN,md,sz16,pre// get byte count from BD
138 6413 // ldf rA,md,sz32 // get address from BD
139 7e01 // bsf Mcerr
140 0006 // ret
141 808d // Mcerr:9: jmp 9b
142 0300 // MblockCh0:done 3 // must wait on MCORE clear HE set HI
143 8051 // jmp GetCh0PC // specific for Channel 0
144 0007 // MGetBDCh0:clrf 0 // make sure flags are clear
145 6b04 // stf rP,ma,pre // new DMA address to load from
146 6516 // ldf rS,md,sz16,pre// get status from buf desc
147 7e06 // bsf McerrCh0
148 4501 // tsti rS,BD_DONE // check for done bit
149 7cf8 // bf MblockCh0 // if zero, not a valid BD
150 6616 // ldf rN,md,sz16,pre// get byte count from BD
151 6413 // ldf rA,md,sz32 // get address from BD
152 7e01 // bsf McerrCh0
153 0006 // ret
154 809a // McerrCh0:9:jmp 9b
155 0007 // DDone: clrf 0 // clear flags
156 6b20 // stf rP,da // address of status word
157 3501 // andni rS,BD_DONE// indicate complete
158 6d32 // stf rS,dd,sz16 // update BD flags (DO NOT FLUSH!)
159 6e36 // stf rN,dd,sz16,fls// update count
160 7f18 // bdf Dcerr
161 4508 // tsti rS,BD_INTR // should we interrupt?
162 7c01 // bf Of
163 0601 // notify 6 // yes, do it
164 1b08 // 0: addi rP,BD_SIZE // bump ptr, assume no wrap
165 4502 // tsti rS,BD_WRAP // are we really at the end
166 7c06 // bf lf // if not, skip the following

```

```

167 0007 // DFirst: clrf 0          // clr flags, could enter here
168 038a //
169 1b04 //
170 6b24 //
171 6333 //
172 7e0c //
173 0006 // 1: ret
174 0700 // Dblock: done 7        // must wait on DSP - clear DE set DI
175 0007 // DGetBD: clrf 0        // make sure flags are clear
176 6b24 // 0: stf rP,da,pre    // new DMA address to load from
177 6536 // ldf rS,dd,sz16,pre // get status from buf desc
178 7e06 //
179 4501 //
180 7cf9 //
181 6636 //
182 6433 //
183 7e01 //
184 0006 //
185 80b9 // Dcerr:9: jmp 9b
186 0970 // Reset: ldi rT,REGPAGE
187 0111 //
188 5201 //
189 4a00 //
190 7c03 //
191 0100 //
192 8051 //
193 80ba //
194 c07b // 0: jsr MFFirst // get address of first BD
195 0f00 //
196 80c6 //
197 c06f // 2: jsr MDone
198 c090 // 3: jsr MGetBDCh0
199 018d //
200 0112 //
201 3907 //
202 4901 //
203 7c02 //
204 078c //
205 80c5 //
206 4902 // 1: cmpeqi rT,C0_LOAD// load command
207 7c0c //
208 6c04 // 4: stf rA,ma,pre
209 008e //
210 0015 //
211 0015 //
212 7804 //
213 6117 //
214 5907 //
215 1f01 //
216 0000 //
217 7e17 //
218 7c01 //
219 80c5 //
220 4903 // 1: cmpeqi rT,C0_DUMP// dump command

```

```

221 7c0b //          bf 1f           // no, try next command code
222 6c00 // 5:       stf rA,ma
223 008e //          mov r0,rN
224 0015 //          lsr1 r0        // byte count -> word count
225 0015 //          lsr1 r0
226 7804 //          loop 4,0
227 5107 //          ld rT,(rL,0)
228 6917 //          stf rT,md,sz32,fls
229 1f01 //          addi rL,1
230 0000 //          yield
231 7e09 //          bsf c0berr
232 80c5 //          jmp 2b
233 4904 // 1:       cmpeq rT,C0_SETCTX// set context command
234 7c02 //          bf 1f           // no, try next command code
235 c0f2 //          jsr CmdCtx
236 80d0 //          jmp 4b
237 4905 // 1:       cmpeq rT,C0_GETCTX// get context command
238 7cca //          bf 9b           // no, ignore
239 c0f2 //          jsr CmdCtx
240 80de //          jmp 5b
241 80f1 //          c0berr:9:jmp 9b
242 0f08 // CndCtx:   ldi rL,CTXPAGE
243 0711 //          revblo rL      // address of contexts
244 018d //          mov rT,rS
245 0112 //          rorb rT
246 39f8 //          andi rT,0xF8    // keep 5 bits (chan * 8)
247 0799 //          add rL,rT      // base += (chan * 8)
248 0799 //          add rL,rT      // base += (chan * 16)
249 0115 //          lsr1 rT      // (chan * 4)
250 0799 //          add rL,rT      // base += (chan * 20)
251 0006 //          ret

```

9.13.4 MCU to DSP transfer

```

252 c056 // MDDma:   jsr Mccb      // get MCORE CCB address
253 5a07 //          st rC,(rL,LS_MC) // save it
254 c07b //          jsr MFFirst    // get MCORE first BD address
255 5b0f //          st rP,(rL,LS_MP)
256 c063 //          jsr Dccb      // get DSP CCB address
257 5a27 //          st rC,(rL,LS_DC) // save it
258 c0a7 //          jsr DFFirst    // get DSP first BD address
259 5b2f //          st rP,(rL,LS_DP)
260 0a00 //          ldi rC,0
261 530f // MDMnxt:   ld rP,(rL,LS_MP) // enter with rC = DSP resid
262 c083 //          jsr MGetBD    // get next MCORE BD
263 5b0f //          st rP,(rL,LS_MP)
264 5d17 //          st rS,(rL,LS_MS)
265 5elf //          st rN,(rL,LS_MN)
266 6c04 //          stf rA,ma,pre // load MCORE DMA address
267 008e //          mov r0,rN      // save count
268 4a00 //          cmpeq rC,0
269 7c08 //          bf 2f
270 532f // MDDnxt:   ld rP,(rL,LS_DP) // enter with r0 = MCORE resid

```

```

271  c0af // jsr DGetBD      // get next DSP BD
272  5b2f // st rP,(rL,LS_DP)
273  5d37 // st rS,(rL,LS_DS)
274  5e3f // st rN,(rL,LS_DN)
275  6c20 // stf rA,da    // load DSP DMA address
276  028e // mov rC,rN    // put DSP count in rC
277  0688 // mov rN,r0    // put MCORE count in rN
278  00d2 // 2:  cmplt r0,rC // calculate loop count
279  7d01 // bt 3f
280  008a // mov r0,rC    // DSP count < MCORE count
281  06a0 // 3:  sub rN,r0    // adjust MCORE count
282  02a0 // sub rC,r0    // adjust DSP count
283  4800 // cmpeqi r0,0 // is trip count 0?
284  7d08 // bt 4f
285  1803 // addi r0,3 // round up from byte count
286  0015 // lsrl r0
287  0015 // lsrl r0 // to full word count
288  7803 // loop 3,0 // clr flags, loop on next 3
289  6117 // ldf rT,md,sz32,pre// LOOP: get 32 bits from MCORE
290  6933 // stf rT,dd,sz32 // LOOP: put 32 bits to DSP
291  0000 // done 0 // LOOP: yield to > priority
292  7c2a // bf MDerr // branch if not count runout
293  4e00 // 4:  cmpeqi rN,0 // MCORE count reach zero?
294  7c16 // bf MDMnz // branch if nonzero
295  008a // mov r0,rC // put DSP resid count in r0
296  5207 // ld rC,(rL,LS_MC) // close MCORE BD
297  530f // ld rP,(rL,LS_MP)
298  5517 // ld rS,(rL,LS_MS)
299  561f // ld rN,(rL,LS_MN)
300  c06f // jsr MDone
301  5b0f // st rP,(rL,LS_MP)
302  563f // ld rN,(rL,LS_DN)
303  4800 // cmpeqi r0,0 // DSP count reach zero?
304  7d03 // bt 0f // branch if zero
305  5117 // ld rT,(rL,LS_MS)
306  4104 // tsti rT,BD_CONT // is MCORE continue bit set?
307  7d07 // bt 1f // yes, don't close DSP BD
308  06a0 // 0:  sub rN,r0 // close DSP BD
309  5227 // ld rC,(rL,LS_DC)
310  532f // ld rP,(rL,LS_DP)
311  5537 // ld rS,(rL,LS_DS)
312  c09b // jsr DDone
313  5b2f // st rP,(rL,LS_DP)
314  0800 // ldi r0,0
315  0288 // 1:  mov rC,r0 // put DSP resid back in rC
316  8105 // jmp MDMnx
317  008e // MDMnx: mov r0,rN // put MCORE resid count in r0
318  5227 // ld rC,(rL,LS_DC) // close DSP BD
319  532f // ld rP,(rL,LS_DP)
320  5537 // ld rS,(rL,LS_DS)
321  563f // ld rN,(rL,LS_DN)
322  c09b // jsr DDone
323  5b2f // st rP,(rL,LS_DP)
324  4504 // tsti rS,BD_CONT // is DSP continue bit set?

```

```

325  7dc8 //          bt MDDnxt
326  561f //          ld rN, (rL,LS_MN) // close MCORE BD
327  06a0 //          sub rN,r0
328  5207 //          ld rC, (rL,LS_MC)
329  530f //          ld rP, (rL,LS_MP)
330  5517 //          ld rS, (rL,LS_MS)
331  c06f //          jsr MDone
332  5b0f //          st rP, (rL,LS_MP)
333  0a00 //          ldi rC,0
334  8105 //          jmp MDMnxt
335  814f //          MDerr:9: jmp 9b

```

9.13.5 DSP to MCU transfer

```

336  c063 //          DMdma:    jsr Dccb           // get DSP CCB address
337  5a27 //          st rC, (rL,LS_DC) // save it
338  c0a7 //          jsr DFfirst        // get DSP first BD address
339  5b2f //          st rP, (rL,LS_DP)
340  c056 //          jsr Mccb           // get MCORE CCB address
341  5a07 //          st rC, (rL,LS_MC) // save it
342  c07b //          jsr MFfirst         // get MCORE first BD address
343  5b0f //          st rP, (rL,LS_MP)
344  0a00 //          ldi rC,0
345  532f //          DMDnxt:   ld rP, (rL,LS_DP)
346  c0af //          jsr DGetBD        // get next DSP BD
347  5b2f //          st rP, (rL,LS_DP)
348  5d37 //          st rS, (rL,LS_DS)
349  5e3f //          st rN, (rL,LS_DN)
350  6c24 //          stf rA,da,pre // load DSP DMA addr, prefetch
351  008e //          mov r0,rN      // save count
352  4a00 //          cmpeqi rC,0
353  7c08 //          bf 2f
354  530f //          DMMnxt:   ld rP, (rL,LS_MP)
355  c083 //          jsr MGetBD        // get next MCORE BD
356  5b0f //          st rP, (rL,LS_MP)
357  5d17 //          st rS, (rL,LS_MS)
358  5e1f //          st rN, (rL,LS_MN)
359  6c00 //          stf rA,ma      // load MCORE DMA address
360  028e //          mov rC,rN      // put MCORE count in rC
361  0688 //          mov rN,r0      // put DSP count in rN
362  00d2 //          2:       cmplt r0,rC     // calculate loop count
363  7d01 //          bt 3f
364  008a //          mov r0,rC
365  06a0 //          3:       sub rN,r0      // adjust DSP count
366  02a0 //          sub rC,r0      // adjust MCORE count
367  4800 //          cmpeqi r0,0     // is trip count 0?
368  7d08 //          bt 4f
369  1803 //          addi r0,3      // round up from byte count
370  0015 //          lsr1 r0
371  0015 //          lsr1 r0      // to full word count
372  7803 //          loop 3,0      // clr flags, loop on next 3
373  6137 //          ldf rT,dd,sz32,pre// LOOP: get 32 bits from DSP
374  6913 //          stf rT,md,sz32 // LOOP: put 32 bits to MCORE

```

```

375 0000 // done 0          // LOOP: yield to > priority
376 7c2a // bf DMerr        // branch if not count runout
377 4e00 // 4:      cmpeqi rN,0 // DSP count reach zero?
378 7c16 // bf DMDnz        // branch if nonzero
379 008a // mov r0,rC       // put MCORE resid count in r0
380 5227 // ld rC,(rL,LS_DC) // close DSP BD
381 532f // ld rP,(rL,LS_DP)
382 5537 // ld rS,(rL,LS_DS)
383 563f // ld rN,(rL,LS_DN)
384 c09b // jsr DDone
385 5b2f // st rP,(rL,LS_DP)
386 561f // ld rN,(rL,LS_MN)
387 4800 // cmpeqi r0,0    // MCORE count reach zero?
388 7d03 // bt Of           // branch if zero
389 5137 // ld rT,(rL,LS_DS)
390 4104 // tsti rT,BD_CONT // is DSP continue bit set
391 7d07 // bt lf           // yes, don't close MCORE BD
392 06a0 // 0:      sub rN,r0 // close MCORE BD
393 5207 // ld rC,(rL,LS_MC)
394 530f // ld rP,(rL,LS_MP)
395 5517 // ld rS,(rL,LS_MS)
396 c06f // jsr MDone
397 5b0f // st rP,(rL,LS_MP)
398 0800 // ldi r0,0
399 0288 // 1:      mov rC,r0 // put MCORE resid back in rC
400 8159 // jmp DMDnxt
401 008e // DMDnz:      mov r0,rN // put DSP resid count in r0
402 5207 // ld rC,(rL,LS_MC) // close MCORE BD
403 530f // ld rP,(rL,LS_MP)
404 5517 // ld rS,(rL,LS_MS)
405 561f // ld rN,(rL,LS_MN)
406 c06f // jsr MDone
407 5b0f // st rP,(rL,LS_MP)
408 4504 // tst irS,BD_CONT // is MCORE continue bit set?
409 7dc8 // bt DMMnxt
410 563f // ld rN,(rL,LS_DN) // close DSP BD
411 06a0 // sub rN,r0
412 5227 // ld rC,(rL,LS_DC)
413 532f // ld rP,(rL,LS_DP)
414 5537 // ld rS,(rL,LS_DS)
415 c09b // jsr DDone
416 5b2f // st rP,(rL,LS_DP)
417 0a00 // ldi rC,0
418 8159 // jmp DMDnxt
419 81a3 // DMerr:9: jmp 9b

```

9.13.6 PPP routine

```

420 0802 // ppprinit:ldi r0,0x2
421 6840 // stf r0,ca      // select CRC-CCITT16
422 c056 // jsr McCb       // get MCORE OCB address
423 c07b // jsr MFFirst     // get BD array pointer of channel
424 c083 // pppm nxt: jsr MGetBD // get first Buffer Descriptor

```

```

425  6c00 //          stf rA,ma      // buffer address
426  008e //          mov r0,rN      // buffer size
427  0900 //          ldi rT,0
428  2101 //          subi rT,1     // RT=0xFFFFFFFF
429  6948 //          stf rT,cs,0   // initialize CRC
430  81b0 //          jmp 2f       // To avoid the wait instruction
431  0400 // 1:         done 4
432  511f // 2:         ld rT,(rL,RECV)
433  7efd //          bsf 1b       // no char, or framing error
434  497e //
435  7fcf //          cmpeqi rT,0x7E
436  7807 // ppprlp:   bf 2b       // not a flag char
437  511f // 4:         loop 7,0
438  497d //          ld rT,(rL,RECV) // check for ESC
439  7d06 //          cmpeqi rT,0x7D // it is an ESC
440  497e //          bt 7f       // check for ending flag
441  7d0d //          bt 9f       // it is the ending flag
442  6949 // 5:         stf rT,cs,run // run CRC on byte
443  6911 //          stf rT,md,sz8 // one byte DMA write
444  7f04 //          bdf 8f       // check for DMA error
445  81c3 //          jmp HOP
446  511f // 7:         ld rT,(rL,RECV) // get the data after the ESC
447  1120 //          xorl rT,0x20 // de-escape the byte
448  81ba //          jmp 5b       // go store it
449  2d10 // 8:         ori rS,BD_MERR
450  81ce //          jmp 1f
451  7e01 // HOP:      bsf UART_WKr // UART watermark has been reached.
452  81c7 //          jmp Of       // Means loop ended with no error
453  0400 // UART_WKr: done 4
454  81b4 //          jmp ppprlp
455  6148 // 0:9:       ldf rT,cs
456  0ef0 //          ldi rN,0xf0
457  0611 //          revblo rN
458  1eb8 //          addi rN,0xB8 // desired final crc value 0xf0b8
459  01ce //          cmpeq rT,rN // compare for equal
460  7d01 //          bt 1f
461  2d40 //          ori rS,BD_CRCE
462  0688 // 1:         mov rN,r0 // size of data transferred
463  c06f //          jsr MDone // finish BD
464  81a8 //          jmp pppmxnt
465  0802 // pppxinit: ldi r0,0x2
466  6840 //          stf r0,ca // select CRC-CCITT16
467  c056 //          jsr Mccb // get MCORE CCB address
468  c07b //          jsr MFFirst
469  81d7 //          jmp pppxnxt
470  c06f // pppxdon: jsr MDone
471  c083 // pppxnxt: jsr MGetBD // don't touch rN,rS,rL
472  6c04 //          stf rA,ma,pre // buffer address
473  008e //          mov r0,rN // buffer size
474  0900 //          ldi rT,0
475  2101 //          subi rT,1 // RT=0xFFFFFFFF
476  6948 //          stf rT,cs,0 // initialize CRC
477  0c7e //          ldi rA,0x7E // flag
478  5c27 //          st rA,(rL,XMIT) // opening flag

```

00000000-0000-0000-0000-000000000000

```
479 780a // pppxlp: loop 10,0
480 6115 // ldf rT,md,sz8,pre// get next data byte
481 497e // cmpeqi rT,0x7E // need to escape escape?
482 7d03 // bt lf
483 497d // cmpeqi rT,0x7D // need to escape escape?
484 7d01 // bt lf
485 7c03 // bf 3f
486 1120 // 1: xori rT,0x20 // modify escaped data (rT = 5E or 5D)
487 0c7d // ldi rA,0x7D // escape
488 5c27 // st rA,(rL,XMIT) // send escape character
489 5927 // 3: st rT,(rL,XMIT) // send data
490 7f01 // bdf UART_WKx // UART watermark has been reached
491 81ee // jmp 0f // loop ended with no error
492 0400 // UART_WKx:done 4 // awake an other channel
493 81df // jmp pppxlp // lm flag is lost when loop is exited
494 6148 // 0: ldf rT,cs // get the 1021 data
495 0111 // revblo rT
496 5927 // st rT,(rL,XMIT) // crc hi byte
497 0111 // revblo rT
498 5927 // st rT,(rL,XMIT) // crc lo byte
499 0c7e // ldi rA,0x7E
500 5c27 // st rA,(rL,XMIT) // ending flag
501 81d6 // jmp pppxdon
```